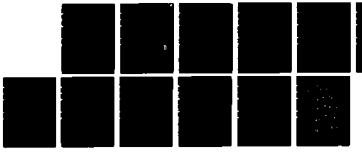
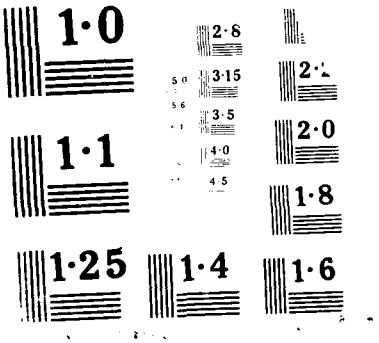
AD-A192 913 RESEARCH IN VLSI COMPUTER SYSTEMS(U) STANFORD UNIV CA 1/1
COMPUTER SYSTEMS LAB J HENNESSY OCT 87
MDA903-83-C-0335

UNCLASSIFIED F/G 12/6 LL




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·8
3·15
3·5
4·0
4·5

COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-2192

Research in VLSI Computer Systems

Technical Progress Report

April 1987 - October 1987

Computer Systems Laboratory

Center for Integrated Systems

This work was supported by the Defense Advanced Research Projects Agency, contracts MDA903-83-C-0335 and N00014-87-K-0828.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-2192

Research in VLSI Computer Systems

Technical Progress Report

April 1987 - October 1987

Computer Systems Laboratory

Center for Integrated Systems

This work was supported by the Defense Advanced Research Projects Agency, contracts MDA903-83-C-0335 and N00014-87-K-0828.

Research in VLSI Computer Systems

Progress Report for April 1987 - October 1987

Center for Integrated Systems
Stanford University
Stanford, California 94305

General Purpose VLSI-Based Multiprocessors
DARPA Contract No. MDA903-83-C-0335
DARPA Order No. 3773-6
Principal Investigator: John Hennessy
Computer Systems Laboratory
Monitored by W. Bandy

Microsupercomputers: Design and Implementation
DARPA Contract No.N00014-87-K-0828
DARPA Order No. 6203-1
R&T Project Code No. 4331685-01
Principal Investigator: John Hennessy
Computer Systems Laboratory
Monitored by W. Bandy and M. Pullen

Abstract

This report summarizes progress in the DARPA funded projects in the areas of multiprocessor architecture and software, and computer-aided design from April 1987 to October 1987. The major areas under investigation have included: analysis and synthesis design aids, high performance chip design, multiprocessor and VLSI computer architectures. The major research problems are introduced and progress is discussed; the Appendix contains a list of published research papers from these projects.

Key Words and Phrases: VLSI, design automation, computer-aided design, special purpose chips, VLSI computer architecture, routing, layout, memory reliability.

This work was supported by the Defense Advanced Research Projects Agency, contracts MDA903-83-C-0335 and N00014-87-K-0828.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Technical Progress

1 Computer-Aided Design

Our activities in CAD tools are focussed on providing the set of tools needed to design the next generation of multiprocessor computer systems. To support this goal, we are working on extending CAD tools upward to help in system level design, as well as downward to support high performance technologies like BiCMOS and ECL. We are working closely with the researchers at Berkeley to merge our tools into their OCT system. We hope this system will make it easier for us and others to use these new tools in a timely manner.

1.1 The Thor Simulation System

The THOR research is broken into three major areas: a production functional simulation environment, incremental simulation research, and parallel simulation research. Each effort is discussed separately in the following sections.

1.1.1 The THOR Environment

The THOR simulation environment has been developing for the past year and has now been released outside Stanford. Copies are currently being used at Berkeley and SRI. The system can run on either VAX or SUN platforms and includes the following capabilities: interactive behavioral simulation (models are 'C' based), graphical logic analyzer, automatic conversion of simple models directly to boolean equations, automatic comparison between THOR and RSIM simulations; and automatic comparison between THOR and the medium tester.

1.1.2 Incremental Simulation

Incremental technique has been proposed for digital simulation to reduce the design validation time through simulation. Incremental simulation implies a simulator that runs in time proportional to the implications of design changes instead of the size of the circuit under simulation. The *incremental* property in the hardware design process is exploited to reduce the number of evaluation of the circuit components, thus achieving run-time performance improvements.

We proposed two incremental simulation algorithms, *incremental-in-space* and *incremental-in-time* algorithms, and implemented them in our *THOR* simulation system. The *incremental-in-space* algorithm simulates all the circuit components affected by design changes since the previous simulation. The *incremental-in-time* algorithm simulates a circuit component only for the simulation time frames when its inputs make different state transitions from the previous simulation run, utilizing the past history of simulation and thus reducing the number of component evaluations to a minimum. Independent of the circuit topology, significant speedups can be obtained in a simulator employing *incremental-in-time* algorithm. This algorithm requires more bookkeeping tasks in maintaining the state for each circuit component (active or inactive) and detecting changes in the state transitions from the previous simulation run. Both algorithms are found efficient, showing up to 30x speedups over conventional event-driven simulation. These two algorithms are comparable to each other: one shows better performance for some circuits over the other, depending on the circuit structure and topology of the circuit under simulation.

1.1.3 Parallel Simulation Study

Two parallel algorithms for logic simulation have been developed and implemented on a general purpose shared-memory parallel machine. The first algorithm is a synchronous version of a traditional event-driven algorithm which achieves speed-ups of 4 to 6 with 8 processors. The second algorithm is a synchronous version unit-delay compiled mode algorithm which achieves speed-ups of 5 to 7 with 8 processors. A third algorithm is being developed that is asynchronous. There are no synchronization locks or barriers between processors and the problems of massive state storage and deadlock have been eliminated. This allows the processors to work independently at their own speed on different elements and at different times. When simulating circuits without feedback, the asynchronous simulation technique varies between 1 to 3 times faster than the conventional event-driven algorithm using 1 processor and achieves 10 to 20% better utilization using 8 processors depending on the circuit.

Staff: B. Alverson, S.Y. Hwang, L. Soule, T. Rokicki, K.Y. Choi, and T. Blank

1.2 Placement and Routing

During the last period we have expanded our effort in placement and routing. In addition to the work on ATLAS, the placement program based on numerical optimization, we began work on Locus, a project to both improve the quality of placement by actually doing the global routing for each placement. This program serves a dual function. It is both an experimental CAD tool, and an experiment in writing parallel applications.

1.2.1 ATLAS

The ATLAS macrocell placement program has now been ported to run on a VAX machine running under UNIX. Instead of using a commercial numerical optimization package, we are now using NPSOL, a set of routines developed here at Stanford. ATLAS now includes: accurate pin positions, translation, rotation, mirroring and also allows rectangles to have only fixed areas with a changeable aspect ratio. In an effort to develop a more efficient solution technique, we are modifying the numerical routines based on our problem specific knowledge. The approach will quickstart the iterative solution process by using the previous solution as the starting seed. Finally, we ran on an industrial example from ZYMOS achieving both a savings in chip area and in solution time.

1.2.2 Locus

The aim of the Locus Project is to replace conventional cost functions used in automatic placement - total wire length or crossing counts - with a much better measure of the goodness of a placement: the actual area. The power of parallel processing is harnessed to increase the speed of the routing. This work is based on standard cell technology because it is possible to get a good estimate of the standard cell routing area by only doing the global routing, and because there are a large number of circuits upon which we can test our ideas. The project two stages: first, to develope a good global router for standard cells, and make it as fast as possible using parallel processingm and second, to create a placement environment which can use the new global router as its cost function.

The first step, a new global router and its parallel implementation is largely complete. The LocusRoute algorithm is based on enumerating a subset of all two-bend routes between two

points, and results in 16% to 37% fewer total number of tracks than the TimberWolf global router for standard cells.. It is comparable in quality to a maze router and an industrial router, but is a factor of 10 times or more faster. Three approaches to parallelizing the router have been implemented: wire-by-wire parallelism, segment-by-segment and route-by-route. Two of these approaches achieve significant speedup - route-by-route achieves up to 4.6 using eight processors, and wire-by-wire achieves from 5.8 to 7.6 on eight processors. These kinds of parallelism are orthogonal, and so their respective speedups will multiply when combined. When these approaches are combined, one problem we will face will be the difficulty of scheduling the different kinds of tasks. The general scheduling problem is NP-complete.

The second step has begun: A placement environment is being developed. It will do the input/output processing for the circuits and the placement optimization. The strategy for optimization is an interesting research question: the new cost function, an actual router, will no doubt have different characteristics than the more typical wire length or crossing count cost functions. The placement system uses the Berkeley OCT database as its input platform. A YAL-to-OCT translator has been written. YAL is the input language used in the recent effort on benchmarking placement algorithms.

Staff: L. Sha, T. Blank, J. Rose, L. Taran.

1.3 High-Level Synthesis and Optimization

The goal of this project is to provide a *front-end* to the design of large scale digital systems. Digital systems are implemented as an interconnection of processors, interfaces and memory. We address both design and implementation issues, devising an interconnection of hardware units that may be mapped automatically to a VLSI implementation.

We envision the entry level description of a system as consisting of two sets of specifications:

- a *behavioral description*, that describes the function of the system to be realized without committing to an implementation, and
- a set of *constraints* on the design. The constraints are related to the technology being used and to the interfacing of the system with the environment.

The second set of specifications allow the designer to specify upper and lower bounds on the time-difference between events corresponding to signal transitions. Since these constraints are tied to an implementation model, they are kept separate from the behavioral specification.

Our design system consists of three major components: **high level**, **logic**, and **physical synthesis**. Synthesis can be seen as a pipeline through each component, which provides a representation that may be processed by subsequent stages. Feedback in the form of estimates on area and timing is provided by the logic and physical synthesis tools to effectively guide optimizations during high level synthesis. High level synthesis, therefore, is pivotal in determining how effectively the spectrum of tradeoffs between performance and area can be explored.

We model **hardware behavior** as a collection of processes. Each process is described by a program in a subset of the C programming language, called HardwareC. Different processes communicate by means of inter-process communication mechanism. The behavior is

represented internally by a graph-based data structure, called the behavioral intermediate form. The functionality of a circuit may be completely specified by a behavioral description in HardwareC. Alternatively, procedure calls may link to external blocks which have already been designed and characterized in terms of area and performance.

The hardware structure is a hierarchical interconnection of modules. Each module is an arbitrary multiple-level combinational or sequential logic circuit, consisting of an interconnection of combinational logic gates, registers, multiplexors, and tri-state drivers. Module interconnection is achieved by wires (unidirectional) and busses (bidirectional). Each module can be processed and optimized by the logic synthesis tools. A macro-cell generator, developed as a part of another project, generates layouts for arbitrary module descriptions.

High level synthesis essentially consists of transforming the behavioral representation into a structural representation which may be physically synthesized. We have built a program that converts HardwareC specification into data-flow and sequencing graphs and achieves the hardware synthesis of the control portion. We have tested the tool on a Motorola 6502, on FRISC (a toy processor used as demonstration for MacPitts) and we are currently trying it on an Intel 8521.

2 Multiprocessor and VLSI Architecture and Software

The activities in this area are aimed at developing scaleable multiprocessors built from high speed (20-100 mips) RISC microprocessors. The research goals include both architecture and software systems for programming these machines. We believe that multiprocessor architectures will require a much tighter coupling to their software systems to achieve acceptable levels of efficiency. Furthermore, we contend that a major stumbling block to developing multiprocessor architectures is the lack of information about how parallel programs behave. We seek to address these problems as we examine alternative multiprocessor architectures.

We have already observed properties in applications that we have studied that dictate certain architectural properties. We plan to continue these efforts, as well as initiate a detailed design and prototype construction.

There are numerous people working in this area. These include: Anant Agarwal, Anoop Gupta, John Hennessy, Mark Horowitz, Paul Chow, Steven Przybylski, and Rich Simoni.

2.1 MIPS-X: A High Performance Microprocessor

The MIPS-X project has focused on the design of a second generation 32-bit RISC microprocessor that built upon the knowledge gained from the results of the Stanford MIPS, Berkeley RISC I and II, and the IBM 801. This project started in the Spring of 1984 and received working silicon in October of 1986. The processor runs at about 16 MHz and has since been integrated into a working prototype board.

Since the completion of the MIPS-X design and the testing of the chip, some further work has been done to modify the chip to get it to run at its specified speed and improve its external inteface. The main goal of recent work has been to document the results of the project and analyze the design methodology used. Four papers and one technical report have been published

or accepted this year as a result. A book that explains the MIPS-X hardware in detail is currently in preparation.

A retrospective look at the MIPS-X design methodology has pointed out several successes and weaknesses. The most important success was the use of the functional simulator as the definition of the behavior of the machine. Verification of the switch-level model that was extracted from the layout was done by running the functional simulator in parallel with the switch-level simulator and using the functional simulator to drive the switch-level simulation. This comparison-mode simulation made it easy to verify sections of the chip with a "working" global definition of the processor. Two weeks were spent in this phase of the project verifying the sections of the chip. After it was felt that the individual sections were working correctly, it was only a day until the full switch-level representation (minus the instruction cache RAM cells) was running Pascal programs. Two more weeks of switch-level simulation were done before the design was shipped to MOSIS. This time can be compared to the final integration of the pieces of MIPS which took about two months. When chips were received, the functional simulator was used to generate the test vectors making it very easy to determine whether the parts were working. Comparison-mode simulation has now been added to the Thor functional simulator being developed at Stanford.

The design and layout of the chip was partitioned according to the functional units. Designers were responsible for all aspects of their piece, starting from the functional simulation, doing the layout and then doing the verification of their part. Up to the point of the final integration, the designers could work independently with the restriction that the signals at the boundary could only be changed after consultation with the other people affected. Verification of the design against the global functional definition meant that partitioning the design was not a problem as evidenced by the short time it took to assemble the pieces and make them work together.

As part of the analysis of the MIPS-X design a detailed chronology of the project was produced to document the history of the project so that the time spent in the different phases of the project could be easily determined. This pointed out a very striking weakness in the design methodology that must occur in all large systems designs. In MIPS-X and in MIPS over half of the design time was spent deciding what to implement even though the implementation methods and methodologies were different. This time was spent evaluating different pipeline organizations, hardware resources, instruction sets and hardware-software tradeoffs. It is important that these iterations be done to find the best solution but there are no tools to help automate and speed up these design loops even though this may be one of the most important phases of the project.

Future work will address the fact that there is really no tool or environment that can help in the early stages of a design when it is important to be able to quickly evaluate alternate solutions. Most of the CAD tools being developed are for the actual design and implementation of a design once it has been specified. If half of the time in a large design is spent specifying what to build and there are no CAD tools to aid in this aspect then it is clear that there is much to be explored in this area.

2.2 Shared-memory Multiprocessor Studies

Shared-memory multiprocessors offer a cost-effective means of achieving high-performance. The shared-memory paradigm also offers a simple programming model. If we would like to build a large-scale shared memory multiprocessor, as a first step we must thoroughly understand shared memory reference patterns. The basic goal of our research is to do just this. We have obtained realistic multiprocessor address traces of large parallel applications. From a better understanding of shared-memory reference patterns in these applications, we can derive good multiprocessor models that can indeed achieve high performance and can be scaled to a large number of processors. For example, our analyses have given us a good understanding of the cache coherence problem. Unlike previous studies that had to make an inordinate number of assumptions about the workload, we can accurately estimate the performance of various cache consistency protocols. We show that the popular snoopy cache schemes cannot be scaled beyond 10 processors or so. Our results indicate that directory schemes are more suited to large scale multiprocessors than snoopy cache schemes because they can be made scalable and still have good performance.

Our study can be roughly divided into three phases. In the first phase (May-June 1987) we developed tools for data collection on a multiprocessor. We modified an existing technique called ATUM for obtaining address traces on uniprocessors to include multiprocessors. This work was done in collaboration with Digital Equipment Corporation, Hudson. The basic idea of this data collection tool was to modify the microcode of a machine to record the addresses touched by an instruction as a side-effect of normal execution. Our current modifications to allow multiprocessor tracing included converting all trace memory accesses to be atomic. We also record all opcodes so that we can identify interesting instructions such as interlocked references. We added a cache in the microcode to filter out repeat opcodes and thus keep the trace size from blowing up. Modifications were also added to allow traces that are an order of magnitude longer than before. A running implementation was developed on a VAX 8350 processor that gives us traces of four processors and are roughly 4 milion in length. We now have traces of parallel applications running under the MACH, VMS, and Ultrix operating systems. These traces are also being made available to the research community in universities. One problem with these traces is that they constitute just three processors and as such is hard to guess how a much larger number of processors will behave. We are currently working on a multiprocessor simulator that builds on top of the VAX T-bit mechanism and can provide traces of a much larger number of processes.

The second and third phases are ongoing ones. The second phase involves characterizing shared-memory reference patterns. Our studies include looking at the amount of sharing in the user and the operating system space, the impact of process migration, and the locality patterns of shared memory reference. In addition, the numbers that we obtain can be used in multiprocessor performance estimation, such as in the performance evaluation of cache consistency protocols, or even to validate earlier methods of evaluation. We show that the operating system does a significant amount of sharing and that a large fraction of interlocked memory accesses also happen in the operating system. Process migration is shown to cause a large fraction of un-shared blocks to appear shared by the processors. Our preliminary findings are summarized in [Agarwal 87a].

The third phase involves looking at specific multiprocessor models and evaluating their

suitability for large scale multiprocessing. We first look at various cache consistency schemes. Our data indicates that bus-based multiprocessors that use snooping cache consistency schemes cannot easily grow beyond 10 or so processors. Directory schemes show promise in large scale shared-memory multiprocessing, especially in systems where the memory is distributed and each processor has some amount of local memory. An evaluation of early directory schemes shows that their performance is only slightly poorer than snoopy cache protocols. However, they cannot be scaled due to the need to do broadcasts, or because the memory directory needs to maintain state associated with each block and the state grows linearly with the number of caches in the system. We introduce scalable directory models that do not require full broadcast capability and need a limited amount of state in the memory directory. Both snoopy cache schemes and directory methods are evaluated in [Agarwal 87b].

In the near future, we hope to get multiprocessor address traces for a large number of processes. We also plan to evaluate in detail scalable directory schemes with these large traces. We should be able to quantitatively compare the performance of earlier multiprocessor models, and our results should help indicate a general direction in which a large-scale multiprocessor can be built.

2.3 Parallel Symbolic Processing: Applications and Architectural Issues

This research focusses on the use of parallelism to speed-up symbolic applications, such as, rule-based expert systems, blackboard systems, constraint-satisfaction problems, design tools for VLSI, etc. We believe that to make significant progress for the above tasks, one needs to use a vertical approach, where all aspects from the application level to the hardware level are addressed concurrently. In particular, at the application level, we have been evalutaing parallel implementations of the OPS5 rule-based language, Assumption-Based Truth Maintenance Systems, and the PROTEAN system. At the hardware level, we are now obtaining and analyzing detailed memory reference traces from the parallel OPS5 application and other VLSI CAD applications to study various synchronization and memory system design issues.

2.3.1 Ongoing Research

In the area of parallel applications, the two main applications that we have been exploring are parallel OPS5 and PROTEAN. OPS5 is a rule-based language that has been widely used to build expert systems. The early work on the parallelization of OPS5 was done at Carnegie-Mellon University (while the author was still there), and included the gathering of detailed traces and simulations to predict the performance on shared-memory multiprocessors. In the past few months, we have actually completed real implementations of the OPS5 on the Encore Multimax and the VAX-11/784 multiprocessors. The results are presented in [Gupta 87], and correspond quite closely to that predicted by the simulations. The report describes the various synchronization bottlenecks that were observed (for example, the single task queue, contention for the hash-table locks for memory nodes), and the solutions that were used to overcome the bottlenecks. The report also describes a number of techniques to build very high performance uniprocessor implementations for rule-based systems. (This work was done jointly with researchers at Carnegie-Mellon University.)

In the parallel implementation of OPS5 discussed above, the parallelization was done by forking multiple processes (using the UNIX fork command) and by making use of shared memory provided by the MACH operating system, and by implementing all the synchronization code

ourselves. An alternative approach to parallelization is to use a parallel programming language to encode the interpreter for OPS5. We present the results of one such implementation of OPS5 in [Gupta 88]. We encode OPS5 in QLISP, which is a parallel dialect of lisp proposed by John McCarthy and Richard Gabriel. We show that QLISP can easily encode most sources of parallelism in OPS5 that had been previously discussed in literature. This is significant because the OPS5 interpreter is the first large program to be encoded in QLISP, and as a result, this is the first practical demonstration of the expressive power of QLISP. Using QLISP we found it very simple to change the granularity at which parallelism is exploited. This makes porting parallel OPS5 to different machines (with different number of processors, with different synchronization and communication overheads) quite easy. Unfortunately, there are no reasonable implementations of QLISP available currently, so we have only been able to run our implementation on a QLISP simulator. (This work was jointly done with Hiroshi G. Okuno, who is a visiting scientist at Stanford.)

PROTEAN, the second application that we have been pursuing, is an application developed at the Knowledge Systems Laboratory (KSL) at Stanford for determining the three dimensional structure of protiens using NMR and NOE data. The application takes as input a set of constraints (for example, distance between molecule A and molecule B is between 3 and 8 angstroms) and tries to figure out all possible configurations that satisfy the constraints. The application consists of a high-level blackboard based AI system that decides the order in which constraints are to be evaluated and a low-level geometry system (GS) that actually applies those constraints. We are currently working on parallelizing the geometry system. The parallel implementation is now running on an 8 processor Encore, and we are working on putting it on a machine that has a much larger number of processors. We would ideally like to have a machine with several hundred processors, but for the time being we will implement it on a 64 processor NCUBE. The notion that we are advancing in the parallel implementation of the geometry system is that of dynamic variable grain parallelism, where the grain at which parallelism is exploited for a particular subcommand is decided at run-time depending on the availability of resources. Initial results show that such a scheme would do significantly better than static approaches. (This work is being done with Andrew Tucker, a graduate student in the Computer Science Department.)

In our trace studies discussed in the previous section, we have also included some symbolioc applications, in particular, the parallel OPS5. One of the things that we observe from the traces is that for very fine-grained parallel applications, like OPS5, the synchronization traffic is very large (about 2% of the references need to access the bus for consistency reasons), and almost any high-performance parallel machine would slow down with that many references going to the bus. This suggests that maybe we should re-examine the application to see how we can change the data structures and synchronization structures to reduce the bus traffic.

2.3.2 Future Directions

We plan to continue to push further in the directions stated above. We are actively seeking new parallel applications to implement and to analyze. One new AI application that we are considering is Assumption-Based Truth Maintenance Systems (ATMSs). ATMSs were developed by Johan deKleer at Xerox PARC and form a common basis for several AI applications. ATMSs are used to keep track of what facts are true under what sets of assumptions, so that the problem solver can reason in multiple worlds at the same time. Another

application that we are considering is the Blackboard systems as proposed by Barbara Hayes-Roth at Stanford. On the architectural front, one limitation of our current memory reference traces is that they correspond to only four processors. We are currently working on a T-bit based simulator that will let us obtain traces for much larger number of processors.

3 Testing

During this period we have continued our work on the design of a low-cost high-speed tester. The basic pin electronics for the tester were designed during the beginning of the year. We created a four-channel test-chip using this design and it was fabricated over the summer. The chips have been fully tested and work extremely well. Each DUT output circuitry contains three timing generators, allowing each pad to be adjusted individually. Two of the timing generators are used to produce the beginning and ending edges for pulse codes (return zero, return one, return tristate, return complement), while the third timing generator is used to set the input sample timing of the DUT pad. The timing generators have about a 500ps resolution, and the chip contains all the circuitry needed to calibrate the pulse generators against a precision external time reference. Using the calibration mode the accuracy of the timing is around 1ns.

In addition to the flexible control of the output timing the chip can select between two output high levels and two output low levels. The input buffer for the DUT pads is simply a clocked sense amp that allows the input theshold to be easily set. The input resolution is about 50mV.

The chip operates at over 33 MVectors/sec in a 2μ CMOS technology, and is small enough to allow one to place 16 pin drives on less than 1/2 a 8mm by 8mm die. We are now looking into compression and decompression methods that can be used to reduce the needed bandwidth or on-chip memory to store the vectors. Our goal is to build the decompressor right onto the pin-drive chips. Our initial work on compression looks good but not great. Compressing the data values seems to give between a factor of 2 to 5 improvement. We are still evaluation different compression methods, and hope to complete the design of the on-chip decompressor sometime this quarter.

Staff: M Horowitz, J. Gasbarro

References

[Agarwal 87a] Agarwal, A. and Gupta, A.
Memory Reference Characteristics of Multiprocessor Applications under MACH.
1987.
Submitted for publication.

[Agarwal 87b] Agarwal, A., Simoni, R., Hennessy, J., and Horowitz, M.
Scaleable Directory Schemes for Cache Consistency.
1987.
Submitted for publication.

[Gupta 87] Anoop Gupta, Charles Forgy, Dirk Kalp, Allen Newell, and Milind Tambe.
Results of Parallel Implementation of OPS5 on the Encore Multiprocessor.
Technical Report CMU-CS-87-146, Carnegie Mellon University, August, 1987.

[Gupta 88] Hiroshi G. Okuno and Anoop Gupta.
Parallel Execution of OPS5 in QLISP.
In *Proceedings of Fourth IEEE Conference on Artificial Intelligence.* March, 1988.

Publications

[Acken 87] Acken, J., Horowitz, M.
A Static RAM as a Fault Model Evaluator.
In *Custom Integrated Circuits Conference*, pages 590-593. IEEE, Portland, OR, May, 1987.

[Agarwal 87a] Agarwal, A. and Gupta, A.
Memory Reference Characteristics of Multiprocessor Applications under MACH.
1987.
Submitted for publication.

[Agarwal 87b] Agarwal, A., Simoni, R., Hennessy, J., and Horowitz, M.
Scaleable Directory Schemes for Cache Consistency.
1987.
Submitted for publication.

[Agarwal 87c] Agarwal, A., Hennessy, J., and Horowitz, M.
An Analytical Cache Model.
1987.
Submitted for publication.

[Agarwal 87d] Agarwal, A., Hennessy, J., and Horowitz, M.
Cache Performance of Operating System and Multiprogramming Workloads.
1987.
Submitted for publication.

[Agarwal 87e] Agarwal, A., Chow, P., Horowitz, M., Acken, J., Salz, A., and hennessy, J.
On-Chip Instruction Caches for High Performance Processors.
In *Advanced Research in VLSI*, pages 1-24. Stanford University, Stanford, CA., March, 1987.

[Carpenter 87] Carpenter, C., Horowitz, M.
Generating Incremental VLSI Compaction Spacing Constraints.
In *Proc. 24th Design Automation Conf*, pages 291-297. ACM, Miami FL, June, 1987.

[Chow 87] Chow, P., Horowitz, M.
Architectural Tradeoffs in the Design of MIPS-X.
In *Proc. 14th Intl. Sym. on Computer Architecture*, pages 300-308. IEEE, Pittsburg, PA, June, 1987.

[Chow 88] Paul Chow and Mark Horowitz.
The Design and Testing of MIPS-X.
In *Advanced Research in VLSI*. MIT, Cambridge, MA, March, 1988.

[Eichenberger 87] P Eichenberger and M. Horowitz.
Toriodal Compaction of Symbolic Layouts for Regular Structures.
In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages . IEEE, Santa Clara, CA, Nov., 1987.

[Gross 87] Gross, T., Hennessy, J., Przybylski, S., and Rowen, C.
Measurement and Evaluation of the MIPS Architecture and processor.
1987.
Accepted for publication.

[Gupta 87] Anoop Gupta, Charles Forgy, Dirk Kalp, Allen Newell, and Milind Tambe.
Results of Parallel Implementation of OPS5 on the Encore Multiprocessor.
Technical Report CMU-CS-87-146, Carnegie Mellon University, August, 1987.

[Gupta 88] Hiroshi G. Okuno and Anoop Gupta.
Parallel Execution of OPS5 in QLISP.
In *Proceedings of Fourth IEEE Conference on Artificial Intelligence.* March, 1988.

[Horowitz 87] Horowitz, M., Chow, P., Stark, D., Simoni, R., Salz, A., Przybylski, S., Hennessy, J., Gulak, G., Agarwal, A. and Acken, J.
MIPS-X: A 20 MIPS Peak, 32-Bit Microprocessor with On-Chip Cache.
Journal Solid State Circuits SC-22(5):790-799, October, 1987.

[Hwang 87] S.Y. Hwang, T. Blank, K. Choi.
Incremental Functional Simulation of Digital Circuits.
In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages . IEEE, Santa Clara, CA, Nov., 1987.

[Miyamoto 87] Miyamoto, J., Horowitz, M.
A Single Chip Functional Tester.
Journal Solid State Circuits SC-22(5):820-828, October, 1987.

[Salz 87] A. Salz, A. Agarwal, P. Chow.
MIPS-X: The External Interface.
Technical Report 87-, Stanford University, 1987.

[Santoro 88] Santoro, M. and Horowitz, M.
A Pipelined 64x64b Iterative Array Multiplier.
In *Intl. Conference on Solid-State Circuits*, pages . IEEE, San Francisco, CA, February, 1988.

[Sha 87] L. Sha and T. Blank.
ATLAS - A Technique for Layout Using Analytic Shapes.
In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages . IEEE, Santa Clara, CA, Nov., 1987.

[Soule 87] L. Soule and T. Blank.
Statistics for Parallelism and Abstraction Level in Digital Simulation.
In *Proc. 24th Design Automation Conf*, pages 588-591. ACM, Miami FL, June, 1987.

[Stark 87] Stark, D., Horowitz, M.
REDS: Resistance Extraction for Digital Simulation.
In *Proc. 24th Design Automation Conf*, pages 570-573. ACM, Miami FL, June, 1987.

[Steenkiste 87a] Steenkiste, P.
LISP on a Reduced Instruction Set Processor: Characterization and Optimization.
PhD thesis, Stanford University, March, 1987.

[Steenkiste 87b] Steenkiste, P. and Hennessy, J.
Tags in LISP: Hardware and Software Approaches.
In *Proceedings of Conference on Architectural Support for Programming Languages and Operating Systems.* ACM/IEEE, Palo Alto, October, 1987.

[Steenkiste 87c] Steenkiste, P. and Hennessy, J.
A Simple Interprocedural Register Allocation Algorithm and its Effectiveness for LISP.
1987.
Submitted for publication.

[Williams 87a] Williams, T., Horowitz, M., Alverson, R., Yang, T.
A Self Timing SRT Division Chip.
In *Advanced Research in VLSI.* Stanford University, Stanford, CA, March, 1987.

[Williams 87b] Ted Williams and Mark Horowitz.
SRT Division Diagrams and their Usage in Designing Custom Integrated Circuits for Division.
Technical Report 87-326, Stanford University, 1987.

END

DATE

FILMED

DTIC

6-88